SOFTWARE ENGINEERING LABORATORY SERIES SEL-81-009

# SOFTWARE ENGINEERING LABORATORY (SEL) PROGRAMMER WORKBENCH PHASE 1 EVALUATION

MARCH 1981

NASA

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

## FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

- NASA/GSFC (Systems Development and Analysis Branch)
- The University of Maryland (Computer Sciences Department)
- Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-81/6091.

Contributors to this document include

| | |
|---|---|
| William J. Decker | (Computer Sciences Corporation) |
| Frank McGarry | (Goddard Space Flight Center) |

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Maryland 20771

## ABSTRACT

This report summarizes the experiences of the Goddard Space Flight Center (GSFC) Code 580 Software Engineering Laboratory (SEL) with the components of a programmer workbench. Phase I of the SEL programmer workbench consists of the design of the following three components: communications link, command language processor, and collection of software aids. A brief description, an evaluation, and recommendations are presented in this document for each of these three components.

## TABLE OF CONTENTS

## LIST OF ILLUSTRATIONS

## LIST OF TABLES

## SECTION 1 - INTRODUCTION

This report summarizes the experiences of the Goddard Space Flight Center (GSFC) Code 580 Software Engineering Laboratory (SEL) with some of the components of a programmer workbench. Programmer workbench is a term which Code 580 personnel apply to an integrated set of software aids made available in a uniform manner on an interactive computer system. Probably the best-known example of a programmer workbench is the Bell Telephone Laboratories' PWB/UNIX (Reference 1).

The SEL programmer workbench is similar in several respects to PWB/UNIX; however, because what was needed was an aid to the development process of the flight-dynamics-type software typical of the Code 580 environment, differences evolved. Specifically, in order for any feature to be included in the SEL programmer workbench, it had to be effective in the development of high-quality flight dynamics software.

The SEL programmer workbench design, as developed by a previous task assignment, specifies the following five major components:

- A high-speed communications link between the SEL development computer (a Digital Equipment Corporation (DEC) PDP-11/70) and the application computer (the Mission and Data Operations (M&DO) IBM S/360-95)
- A shared supervisor task on the PDP-11/70 which manages the task of each individual session and queues all transmissions on the communications link
- A command language processor to provide the interface between the user and the session task
- A file librarian system to map the command language file specifications into the actual PDP-11/70 or IBM S/360-95 file designations and to control the use of public shared-access libraries

- A collection of software aids useful to flight dynamics software development

The designs of three of these five components have been refined enough to be evaluated at this time. These three are (1) the communications link, (2) the command language processor, and (3) the software aids. The evaluations given in this document attempt to describe the strengths and weaknesses of each and, where possible, indicate new directions that might be taken when further refining of the design is complete.

Sections 2, 3, and 4 present a brief description, an evaluation, and recommendations for the communications link, software aids, and command language processor, respectively.

## SECTION 2 - COMMUNICATIONS LINK

### 2.1 INTRODUCTION

The communications link is the component of the SEL programmer workbench that enables users on the development computer (a PDP-11/70) to submit jobs to be processed on the applications computer (an IBM S/360-95). The separation of the development and applications areas has the advantage of reducing the scheduling and priority allocation conflicts that arise when these areas must share resources. However, total isolation of these areas is not practical, especially in the later stages of software development when tests must be performed on the applications machine and error correction performed on the development machine. The communications link provides an efficient alternative to shifting the entire development effort to the applications machine.

### 2.2 COMMUNICATIONS LINK DESCRIPTION

The communications link enables users of the PDP-11/70 to submit jobs to the IBM S/360-95 and to receive output from the completed jobs. The link hardware consists of a DQ11 Synchronous Serial Interface and a dedicated 9600 baud line. The link software (the RJE Program) emulates an IBM 3780 Remote Job Entry (RJE) terminal.

The RJE Program was written for the PDP RSX-11D operating system by GSFC Code 934. The program was converted to the RSX-11M operating system for Code 580 by Systex, Incorporated and became operational within the SEL in June 1980. Since that time, the program has seen some limited by steady use.

### 2.3 EFFECTIVENESS AND USE

In one sense, this type of communications link can be considered to be a one-way link; i.e., the PDP-11/70 users can task the IBM S/350-95, while the S/360-95 users cannot task the PDP-11/70. However, this is not a limitation upon the intended purpose of the PDP-360 link, because there are few times when an applications environment generates tasks for the development effort. In other words, software "flows" from the development area to the applications area, and this fact is reflected in the RJE link capabilities. A more complex communications setup is therefore not required.

One telling observation can be made at this time. After almost 1 year of availability to Code 580 development projects, the communications link has not been demonstrated to be critically needed. If it were not available, no current or planned project would be stopped or seriously delayed. All projects using the RJE Program have alternate (although slower and less convenient) methods of submitting jobs to the IBM S/360-95.

The reasons for the lack of a critical role for the communications link in the Code 580 development efforts are not readily apparent. Since the RJE Program is easy to use and functions reliably, user dissatisfaction does not seem to be the cause. A more likely reason is the relative newness of the idea. Project planners may be unaware of the RJE capability or unfamiliar with the ways in which it can be used to facilitate the development-area-to-applications-area transition.

## 2.4 CONCLUSIONS AND RECOMMENDATIONS

To date, the RJE communications link has fulfilled its purpose in demonstrating the feasibility of a connection between the PDP-11/70 and the IBM S/360-95. The RJE Program has also shown that communications between a development machine and an applications machine can be effective when only the development machine can generate tasks. The importance of this second finding lies in the fact that if a more complex communications link is not required, a limit is placed upon the complexity (and, hence, the cost) of the link software.

It is recommended that some effort be made to include the use of the RJE Program in the preliminary plans for a selected SEL project so that it can be fully integrated into the complete software development process from the start. Thus, the full impact of the carefully planned use of the RJE link capability could be assessed.

No changes are contemplated to the RJE Program at the present time. This is due primarily to the current simplicity and ease of use of the RJE capability. Another consideration, however, is the uncertainty surrounding the details of the components and structure of a proposed new Flight Dynamics System computing facility. The design of the new system could possibly eliminate the need for a separate communications link for the programmer workbench through the incorporation of multipurpose links.

# SECTION 3 - SOFTWARE AIDS

## 3.1 INTRODUCTION

The concept of the programmer workbench calls for an integrated set of software aids which can be applied to the Code 580 software development process. The term "software aid" as used here includes development tools and utilities from any source. For example, many basic software aids are usually supplied by the computer vendor along with the hardware. These aids include file manipulation utilities and compilers for the major high-order languages. Other more complex but still general-purpose aids, such as data base management systems or word processing software, are available from independent software vendors.

However, experience in the SEL indicates that the greatest success has been achieved with tools or utilities developed in-house to satisfy applications specific to the Code 580 environment.

## 3.2 GSFC CODE 580 ENVIRONMENT

The Code 580 computing environment can be described in terms of the size and the type of software development projects and the application areas served by the development.

Code 580 software development results in software systems that range in size from 5,000 to 120,000 lines of code. A typical (average) system has 40,000 lines. When possible, a high-order language (typically FORTRAN) is used. The development is carried out primarily in an interactive environment on both PDP-11/70 and IBM S/360 computers.

The software can be characterized as scientific application systems with little or no real-time or near-real-time requirements. Attitude determination and control systems require software to access large data bases and to perform flight dynamics analysis. Orbit determination and control systems require celestial mechanics software that is mainly mathematical and algorithmic. Spacecraft maneuver planning requires mathematical and algorithmic software that models a particular vehicle's physical and dynamic characteristics. Mission planning software is the generalized maneuver planning software that is used to evaluate vehicle performance while the total mission is still in its definition phase.

## 3.3 CRITERIA FOR SOFTWARE AIDS

The following two lessons have been learned about what makes a utility or tool useful to SEL users:

- A software aid is more effective when it is simple.
- The set of software aids must be an integrated set.

### 3.3.1 SIMPLICITY

Experience within the SEL tends to indicate that a simple tool or utility achieves wide and long lasting acceptance more often than a complex tool or utility. Simplicity here means that each aid should have a single purpose, with a small number of options. The options should provide flexibility of function for the aid but should not add unrelated capabilities to it.

The interaction with the user is thus limited to prompting for information needed to perform one function. If the user makes an error, it is more likely to be detected as an error because it cannot be interpreted as a request for an alternate function.

### 3.3.2 INTEGRATED SET

An integrated set of software aids is achieved when the aids are invoked with a common syntax and when the range of capabilities is adequate to allow the user to perform all required actions. A uniform syntax is important to the user, since it results in a shorter learning period and a lower error rate after the syntax is learned.

Section 4 of this document describes a proposed syntax and list of commands. These commands are representative of current capabilities within the SEL, but they do not represent the only possible list.

The selection of software aids for inclusion in the programmer workbench will continue until well after the introduction of the workbench into the SEL environment.

## 3.4 CONCLUSIONS

In conclusion, the following can be said about software aids for the SEL programmer workbench:

- The tools and utilities to be selected should perform a single function.
- The common command syntax implied by the programmer workbench concept will in itself be an aid to users.
- The list of software aids included in the programmer workbench is expected to evolve with time.

# SECTION 4 - COMMAND LANGUAGE PROCESSOR

## 4.1 INTRODUCTION

The command language processor is the component of the programmer workbench that ties together all components into a useful whole. The processor interprets the user's typed commands and invokes the particular component of the workbench required to perform the requested function.

The effectiveness of the programmer workbench concept will depend heavily on the user's acceptance of the system, and a well-structured, easy-to-learn-and-use language will contribute to user satisfaction.

The following subsections describe the proposed command language, recommend some additions to the language, and present some arguments in support of continued in-house development of the command language processor (as opposed to the use of software from other workbench projects).

## 4.2 SEL COMMAND LANGUAGE DESCRIPTION

The syntax and lexicon of the SEL, as developed in the previous task assingment, are given in Figure 4-1 and Table 4-1, respectively. The language is structured to take advantage of the processor-defined defaults whenever possible. For example, if the user enters

```
EDIT MODULE
```

the command language processor will assume that the file MODULESRC.FPP is to be edited, since the default type of file content is source code (SRC) and the default language is structured FORTRAN (FPP). Of course, the user can override these defaults if desired, but the defaults have been chosen to minimize this need in the Code 580 environment.

```
 2  BASELINE[ pfile][,level]
 4  BACKUP[ library]( TO |>)filename
 6  CALCULATE[ expression]
 8  CALL[ pfile][,CREATE|,EDIT]
10  CHANGE /string-1/[string-2]/[ IN (pfile|library|ALL)][,LIST]

12  COMPILE[ pfile][,REMOTE]
14  COMPARE[ pfile][ WITH sfile][( TO |>)output]
16  CONTROL[ pfile][,CREATE|,EDIT][,REMOTE]
18  COPY[ pfile]( TO |>)sfile[,N-n]
20  CREATE[ pfile][/subfile][,DELETE]

22  DATE[mm/dd/yy][,format]
24  DEBUG[ pfile][,CREATE|,EDIT]
26  DELETE[ pfile][/subfile[,subfile]...]
28  DIRECT[ pfile| library| ALL][,format]
30  EDIT[ pfile][( TO |>)sfile]

32  EXECUTE[ pfile]
34  EXIT[,RESTART]
36  FIND /string/[ IN (pfile|library|ALL)][,LIST]
38  GESS[ pfile][,REMOTE]
40  GESSDOC[ pfile][( TO |>)sfile][,option]

42  HELP[ command]
44  INSTALL[ pfile][ IN library]
46  LINK[ pfile]
48  LIST[ pfile][/subfile[,subfile]...]
50  LOAD[ pfile][,CREATE|,EDIT][,USER|,TASK]

52  PFILE[ pfile]
54  PRINT[ pfile][/subfile[,subfile]...][/printer]
56  REGEN pfile/version
58  RENAME pfile( TO |>)sfile
60  RESTORE[ library] FROM filename

62  RETRIEVE filename[,filename...]
64  RUNOFF[ pfile][( TO |>)output]
66  SCAN[ pfile][/subfile]
68  SDOC[ pfile][,level]
70  SIZE[ pfile][,SUBFILE]

72  STATUS jobname
72  SUBMIT clist[,target]
76  SYNCH[ pfile][ WITH sfile][( TO |>)stpfile]
78  TEST[ pfile][,CREATE|,EDIT]
80  TIME[time[,date]][/format][,STOPWATCH|,SESSION]

82  TRACE[ pfile][,CREATE|,EDIT]
84  TRANSMIT clist[,target]
86  UPDATE[ pfile],scriptfile
88  XREF[ pfile][,options][( TO |>)output]
```

Figure 4-1. Command Language Syntax (1 of 2)

** NOTATION KEY **

| NOTATION | | MEANING |
|---|---|---|
| UPPER CASE | | REQUIRED SPELLING |
| LOWER CASE | | USER-SUPPLIED INFORMATION |
| SQUARE BRACKETS | [ ] | OPTIONAL INPUT |
| PARENTHESES | ( ) | OPTIONS WITHIN AN OPTION |
| VERTICAL BARS | ¦ | ONE OF SEVERAL IS TO BE SELECTED |

Figure 4-1. Command Language Syntax (2 of 2)

Table 4-1. Command Language Lexicon (1 of 3)

| Command | Description |
|---|---|
| BASELINE | Produce a baseline tree chart with the specified file as the root, extending for a specified number of levels |
| BACKUP | Copy the working (or other specified) library to a packed file |
| CALCULATE | Enter calculator mode, evaluate an expression |
| CALL | Use (or create or modify) a command list to execute a module, performing necessary compiling and task building |
| CHANGE | Global edit function to change (and list) all occurrences of specified string in a file or a library (see FIND) |
| COMPILE | Precompile, compile module (optionally, on target system) |
| COMPARE | Compare two files, list differences, optionally produce SLP editor script |
| CONTROL | Generate command list (as used by CALL or SUBMIT) |
| COPY | Produce copy of the file with new generic name, version = 1 |
| CREATE | Call EDI to create new text file (defaults to ______SRC.FPP, but also used for GESS, test files, documentation, and others) |
| DATE | Display current date in selected format (also used as a format converter; e.g., calendar day to Julian day) |
| DEBUG | Specify debug mode for execution of module (see CALL) |
| DELETE | Mark generic name (or specific subfile) for deletion |
| DIRECT | Produce directory listing of working (or other specified) library, with various formatting/ processing options |
| EDIT | Call EDI to edit file (see CREATE); may also perform copy function prior to editing (see COPY) |

Table 4-1. Command Language Lexicon (2 of 3)

| Command | Description |
|---|---|
| EXECUTE | Task build and execute module, compiling if necessary; unlike CALL, does not use command list |
| EXIT | End session, delete files marked by DELETE, optionally restart session |
| FIND | Global search function to list all occurrences of specified string in module or library (see CHANGE) |
| GESS | Process GESS source, optionally on target system (similar to COMPILE) |
| GESSDOC | Extract system description data from GESS source files |
| HELP | Print description, format, defaults of specified command |
| INSTALL | Copy specified module source into controlled library |
| LINK | Create task (compile if necessary) from module |
| LIST | List specified file on terminal (see PRINT) |
| LOAD | Use (or create or modify) command list to compile module and install the ------SRC.OBJ file into the object library |
| PFILE | Display (or specify) the primary default module name |
| PRINT | Print specified file on printer (see LIST) |
| REGEN | Regenerate specified intermediate version of controlled source from original source and SLP editor script |
| RENAME | Rename specified generic module file or subfile |
| RESTORE | Copy working (or other specified) library from backup packed file (see BACKUP) |
| RETRIEVE | Retrieve target system output data sets to programmer workbench |
| RUNOFF | Call text processor for format module onto output device/file |
| SCAN | Call fast-look editor to examine listings, output files |

Table 4-1. Command Language Lexicon (3 of 3)

| Command | Description |
|---|---|
| SDOC | Extract prologue and program design language (PDL) from module source files and from dependent modules, as required (see BASELINE) |
| SIZE | List size characteristics of module or subfile |
| STATUS | Return status of specified job on target system |
| SUBMIT | Queue command list and files for submission to target system |
| SYNCH | Produce SLP script to convert one file into second file (see COMPARE) |
| TEST | Specify test mode for execution of module (see CALL); use temporary version of module (cf. PANVALET) |
| TIME | Display current time in optionally specified format; also used as format converter (see DATE) |
| TRACE | Specify trace mode for execution of module (see CALL) |
| TRANSMIT | Use (or create or modify) command list to move files between the programmer workbench and target system |
| UPDATE | Use SLP script to update controlled source (see SYNCH) |
| XREF | Create specified type of cross reference from module or from working (or specified) library |

A further extension of the default definition idea is to extend the concept to the module name itself. For example, if the user enters

```
EDIT MODULE
COMPILE
```

the compiler selected will be the structured FORTRAN compiler and the input to the compiler will be the file MODULESRC.FPP.

The use of defaults is quite common in interactive command languages and results from a desire to reduce the number of user keystrokes and, therefore, the chance for error. Another consideration is the relative speed with which a command is typed, compared with the machine response time and the user's thought processes. The command language processor answers this problem with multiprocess commands such as CALL (COMPILE + LINK) and LOAD (COMPILE + INSTALL). In this way, common sequences are collected into one command.

## 4.3 USEFUL FEATURES ABSENT FROM THE COMMAND LANGUAGE PROCESSOR DESIGN

Although much thought was given to command ease of use, the following two features which should be included in the command language processor design were omitted:

- Stored command sequence file processor
- Last-command-recall capability

The stored command sequence processor is a utility that reads a specified file containing command language statements and executes them as if they were entered by the user. Such a processor is available in almost all command languages with various levels of sophistication. This feature enables users to control quite complex and often unique processes with an absolute minimum of keystrokes.

With the last-command-recall feature, the user can recover the last typed command for modification and/or resubmission.

A series of commands containing only small differences can thus be executed quickly. The primary benefit occurs when the user can recall a command after a syntax error is detected. In this case, the user need only correct the part of the command containing the error before continuing.

## 4.4 IN-HOUSE DEVELOPMENT CONSIDERATIONS

The choice of developing a command language processor in-house for the SEL programmer workbench (rather than using software from other workbench projects) has the advantage of close control of the language capabilities, which is necessary in a research environment.

In-house development allows the addition of a monitoring feature to the processor. This monitor can extract information about the commands that are processed (e.g., command use frequency, error rates, or execution success/failure). These statistics can be used by management to monitor progress in a particular software development project.

The statistics can also be used by programmer workbench developers to evaluate the system's effectiveness and performance. Commonly used command sequences can be detected and incorporated into the language as new commands, and frequently used com .nds can be streamlined into a simpler syntax. Language elements which are not used or are determined to not be effective in developing flight dynamics software may even be removed from the language, thus reducing the confusion that a cluttered language can cause.

## 4.5 RECOMMENDATIONS

Further work needs to be done on the command language processor design. In particular, work should be concentrated in the following areas:

- More detailed design of the default definition rules, especially in the transition from single-module commands (EDIT, COMPILE) to multimodule commands (LINK, EXECUTE)
- Establishing priorities for a staged implementation of the processor
- A continuing review of the particular commands to be included

## REFERENCES

1. Dolotta, T. A. and Mashey, J. R., "An Introduction to the Programmer's Workbench," Proceedings of the Second International Conference on Software Engineering, October 13-15, 1976

## BIBLIOGRAPHY OF SEL LITERATURE

Anderson, L., "SEL Library Software User's Guide," Computer Sciences-Technicolor Associates, Technical Memorandum, June 1980

Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development for Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

Basili, V. R., "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1980

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

Basili, V. R., and J. Beane, "Can the Parr Curve Help with the Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and T. Phillips, "Validating Metrics on Project Data," University of Maryland, Technical Memorandum, December 1981

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Church, V. E., "User's Guides for SEL PDP-11/70 Programs," Computer Sciences Corporation, Technical Memorandum, March 1980

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Perricone, B. T., "Relationships Between Computer Software and Associated Errors: Empirical Investigation" (paper prepared for the University of Maryland, December 1981)

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Software Engineering Laboratory, SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

--, SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

--, SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

--, SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu, D. S. Wilson, and R. Beard, September 1977

--, SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

--, SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, January 1978

--, SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

--, SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

--, SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson, B. Chu, and G. Page, September 1978

--, SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

--, SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer, November 1978

--, SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

--, SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

--, SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

--, SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, S. R. Waligora, and A. L. Green, August 1979

--, SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and F. E. McGarry, September 1979

--, SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

--, SEL-80-001, Configuration Analysis Tool (CAT) Functional Requirements/Specifications, F. K. Banks, C. E. Goorevich, and A. L. Green, February 1980

--, SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker, C. E. Goorevich, and A. L. Green, May 1980

--, SEL-80-003, Multimission Modular Spacecraft Ground Support System (MMS/GSS) State-of-the-Art Computer System/Compatibility Study, T. Welden, M. McClellan, P. Liebertz, et al., May 1980

--, SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

--, SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

--, SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

--, SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

--, SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

--, SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, F. E. McGarry, et al., September 1981

--, SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, G. Page, et al., September 1981

--, SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

--, SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

--, SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

--, SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker, A. L. Green, and F. E. McGarry, March 1981

--, SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

--, SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

--, SEL-81-012, Software Engineering Laboratory, G. O. Picasso, December 1981

--, SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

--, SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

Turner, C., G. Caron, and G. Brement, "NASA/SEL Data Compendium," Data and Analysis Center for Software, Special Publication, April 1981

Turner, C., and G. Caron, "A Comparison of RADC and NASA/SEL Software Development Data," Data and Analysis Center for Software, Special Publication, May 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977